ThePoorEngineer

WORKING HARD TO BE LAZY

MENU

COPTER ANGLE CONTROL (RELATIVE)

Posted on March 11, 2018

Table of Contents

Alright, we are finally at the interesting part. This chapter is really long so I suggest that you take a break after reading each section. In this chapter, we will be applying our control system to levitate our copter arm to the angle that we want. Unlike the motor controller in the previous chapter where the system of equation is first order (only a first order derivative present), You will see that a Proportional only controller is not able to stabilize a second order system (second order derivative present) so we will need to implement a Proportional and Derivative (PD) or Proportional, Integral and Derivative (PID) controller. Here are the codes that I have used to run the control system:

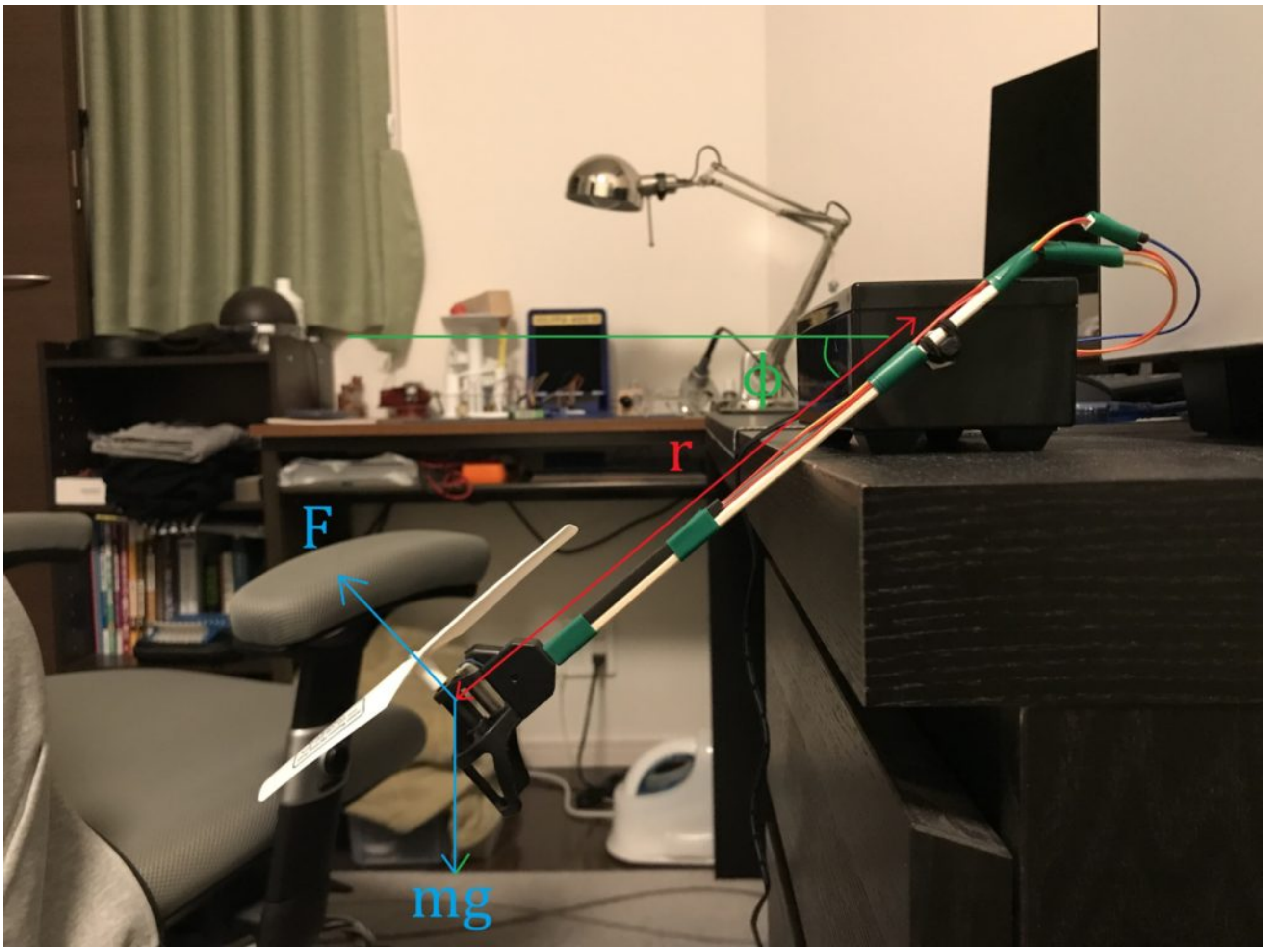
- **Python GUI Source Code for Copter Angle Control**
- **Arduino Source Code for Copter Angle Control**
- **Arduino Library for PID Control**

The relative control here means that we will be controlling the angle relative to the box, and not to the world. As such, when when we rotate the box, the copter arm will rotate follow accordingly as well. On the other hand, the next chapter on absolute control will show that the copter arm stays at where it is regardless of how you rotate the box.
In case you have not used Python before, you can follow **this post** to install a Python IDE to run the code. If you are looking to understand the code, you should read chapter 4 and 5 on how to **plot Serial data** and **creating a GUI** with Python. Here's the final result first to catch your interest 🙂

Propeller Arm Angle Control (PID) with Arduino and Python

Mathematical Model

Before we start writing out the mathematical model, let us take a look at the free body diagram of the set up.



COPTER FREE BODY DIAGRAM

I apologize for the mess in the picture. That is my room, and I did not have a cardboard for the background. Anyway, from the freebody diagram above, we can derive the following equations given that the summation of moment about the pivot is 0.

$$J\ddot{\phi} = r(F - mgcos(\phi)) \text{ ———- } (1)$$

where:
J is the moment of inertia of the propeller and the motor shaft (everything that is spinning)
ϕ is the angle of the copter arm relative to the horizontal (-ve below, +ve above)
$\ddot{\phi}$ is the angular acceleration of the copter arm (not the motor!)
r is the length of the moment arm
F is the Force by the propeller
m is the mass of the copter (including the arm)
g is the gravitational constant

In actual fact, the chopstick and the wires contribute to the weight as well so the center of mass of the whole system should be shifted up the arm. However here, to make things simple, we assume that the moment arm is the same for both the force (F) and the weight of the copter (mg). In other words, both forces act at the same point on the copter.

Alright, we are done with this part. Let’s move on to get the equation of motion in discrete time. The above equation is actually the equation of motion in continuous time.

Equation of Motion

From equation 1 above, let $\dot{\phi} = \theta$ so $\ddot{\phi} = \dot{\theta}$. Then,

$$\frac{J}{r}\dot{\phi} = F - mgcos(\phi)$$

Assuming that the force by the propeller (F) is proportional to our motor input (this is a very huge assumption that is largely untrue but you will see that it works nonetheless. That is the power of the control system. Ain't it just amazing?!),

$$F = AI_{pwm}$$

where
A is the constant of proportionality relating the force and the motor PWM input
I_{pwm} is the motor PWM input

Substituting the Force from the original equation,

$$\frac{J}{r}\dot{\phi} = AI_{pwm} - mgcos(\phi)$$

Converting the above to discrete time,

$$\frac{J}{r}\frac{(\theta[n]-\theta[n-1])}{t} = AI_{pwm}[n-1] - mgcos(\phi[n-1]) \text{ ———- } (2)$$

Since $\dot{\phi} = \theta$,

$$\frac{\phi[n]-\phi[n-1]}{t} = \theta[n-1] \text{ ———- } (3)$$

Substituting equation 3 into 2 and simplifying, you should get the following equation:

$$\phi[n] - 2\phi[n-1] + \phi[n-2] = t^2\gamma I_{pwm}[n-2] - t^2\beta cos(\phi[n-2]) \text{ ———- } (4)$$

where
$\gamma = \frac{Ar}{J}$
$\beta = \frac{mgr}{J}$

Now we finally have our equation of motion in discrete time so let us do a little bit of analyzing. At time infinity, assuming that the system is stable (response converges to a value),

$$\phi[n] = \phi[n-1] = \phi[n-2] = \phi[\infty]$$

Therefore equation 4 turns into:

$$\phi[\infty] - 2\phi[\infty] - \phi[\infty] = t^2\gamma I_{pwm}[\infty] - t^2\beta cos(\phi[\infty])$$

Here, let us assume that our control input to the motor ($I_{pwm}[\infty]$) is a constant so we can rewrite it simply as I_{pwm}. Then, simplifying the above equation will yield the following:

$$\gamma I_{pwm} = \beta cos(\phi[\infty])$$

Here, I_{pwm} is a parameter that we can control as we wish. If we let $I_{pwm} = \frac{\beta}{\gamma} cos(\phi_{desired})$, we will end up with:

$$\phi[\infty] = \phi_{desired}$$

meaning that the system will eventually reach the desired angle that we set! In fact, $I_{pwm} = \frac{\beta}{\gamma} cos(\phi_{desired})$ is the required input to get the system to any desired angle. However, our model is never perfect so even if you have accurate values of β and γ, it is usually impossible to get the system to every single possible angle accurately. In addition, we have not yet discuss about the system response. The system will reach the desired angle eventually but that may be 2 or 5 seconds later. The control system provides us with a way to get the copter arm to our desired position quickly as you will see later. For now, keep in mind that $I_{pwm} = \frac{\beta}{\gamma} cos(\phi_{desired})$ is the input that we need to estimate the input to the controller in order to achieve a certain angle for the copter arm.

Calibration

From the equation of motion (equation 4 in red), you can see that there are 2 variables that are unknown thus we have to determine its value through experiments. 2 unknowns mean that we need 2 equations in order to solve for their values. The first equation is actually already written above,

$$I_{pwm} = \frac{\beta}{\gamma} cos(\phi_{desired})$$

When we set a certain value for the motor PWM, the copter arm will rise to a certain angle. By measuring this angle, we can get the first equation to solve for the 2 unknowns. Although we cannot solve for both unknowns using just the above equation, it is possible to determine the value of \Large{\frac{\beta}{\gamma}} since this is actually just a constant. I did some simple measurement and below are my results. You will need to first calibrate the potentiometer to know what is the respective angle from the Arduino's `analogRead()`.

PWM	Angle	Angle (degree)	β/γ
70	430	-22.8901734104	75.9835421575
65	417	-26.2716763006	72.4875791583
60	380	-35.8959537572	74.0664801391
		AVE	74.179200485

You will most certainly get a different value from me but the process to do it is the same. With the value of \Large{\frac{\beta}{\gamma}}, we can now set the copter to approximately get to any position that we desire.

Moving on, the 2nd equation to solve for the 2 unknowns is actually a little more tricky. We have to look at the system response to an input in order to determine it. This requires some background on solving 2nd order difference equations which I am not going to do here. Instead, I will just write the method to solve it. If you are interested in how this method come about, you can take a look at this **page** or this **youtube video** for more information.

In order to get the second equation, let us assume that we have a proportional and derivative (PD) controller present. At this point, I will just give the equation of the controller but there will be a more detailed explanation of it in the PD controller section. Let the control input be given by:

$$I_{pwm}[n-2] = \frac{\beta}{\gamma}cos(\phi_{desired}) + K_p(err[n-2]) + K_d\left(\frac{err[n-2]-err[n-3]}{t}\right) \text{ ———- (5)}$$

where

$err[n-2] = \phi_{desired} - \phi[n-2]$

Error (represented as err in the equation) is the difference between the desired angle and the current angle. Notice that the first term in equation 5 is the term that is required to set the copter arm to our desired angle as shown previously (we already have the value of the ratio of beta and gamma through the calibration above). The second term that include K_p is the proportional term (this is the same as in **motor speed control** as well), and the third term with K_d is the derivative term. Together, this is the PD controller.

Substituting equation 5 to our equation of motion (equation 4) and simplifying, you will get the following equation:

$$\phi[n] - 2\phi[n-1] + (1 + t^2\gamma(K_p + \frac{K_d}{t}))\phi[n-2] - t\gamma K_d\phi[n-3] = (t^2\gamma K_p)\phi_{desired} \text{ ———- (6)}$$

Considering only the homogeneous equation,

$$\phi[n] - 2\phi[n-1] + (1 + t^2\gamma(K_p + \frac{K_d}{t}))\phi[n-2] - t\gamma K_d\phi[n-3] = 0$$

We can then solve for the natural frequencies of the system by converting the above to a polynomial and solving for the roots of the equation. The polynomial is given by:

$$\lambda^3 - 2\lambda^2 + (1 + t^2\gamma(K_p + \frac{K_d}{t}))\lambda - t\gamma K_d = 0$$

This is a 3rd order polynomial in the form of:

$$a\lambda^3 + b\lambda^2 + c\lambda + d = 0$$

where

$$a = 1$$
$$b = -2$$
$$c = 1 + t^2\gamma(K_p + \frac{K_d}{t})$$
$$d = -t\gamma K_d$$

Solving for the roots of a polynomial is easy, but this time round, we have the roots, but need to solve for the value of gamma that gives the value of the root. This is an iterative process that is not meant to be done by humans. So I wrote a script in python to do this job.

Before I show you script, let me explain a little about what is happening here. The natural frequency (roots of the polynomial) actually dictates the response of the system. One measurement that we can do through experiments is the period of oscillation. If the system shows an oscillatory response, this means that at least one of the roots of the polynomial above is complex (meaning that it has both a real and an imaginary part). The imaginary part of the root is actually the frequency of oscillation which can be related to the period of oscillation. As such, when we measure the period of oscillation, we are actually determining the value of the imaginary part of the most prominent complex root (given that the roots are far apart from each other).

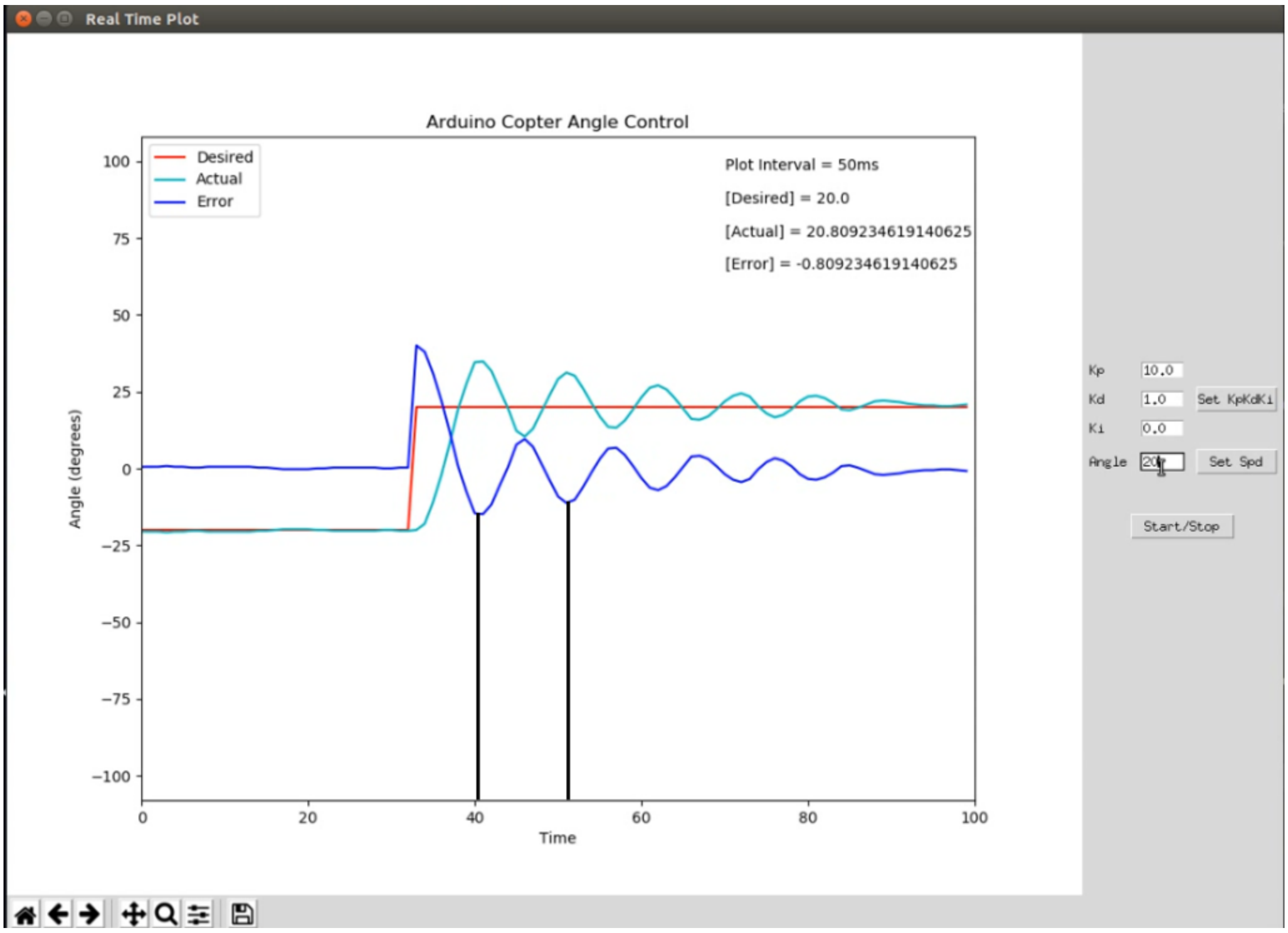
Here's the promised python script to calculate the value of γ. This program iterates through multiple test values to determine the value of gamma which produces a root that is closest to value that you have measured

```
import numpy as np


t = 0.001    # [s]
Kp = 10.0
Kd = 1.0

testRange = np.arange(0, 30, 0.01)
periodOfOscillation = 0.55
differentialStep = 1
minDiff = 99999
actualGamma = None

# determine gamma
for gamma in testRange:
    x3 = 1
    x2 = -2
    x1 = 1 + (t**2) * gamma * (Kp + Kd/(differentialStep*t))
    c = -(t/differentialStep) * gamma * Kd
    coeff = [x3, x2, x1, c]
    roots = np.roots(coeff)
    index = np.argmax(np.absolute(roots))
    diff = abs(2.0*np.pi/periodOfOscillation - np.imag(roots[index]))
    if (diff < minDiff):
        minDiff = diff
        actualGamma = gamma

print("Actual Gamma: " + str(actualGamma))
```

In the program, t is the time interval in seconds. All you have to do is to input the period of oscillation that you have measured during the experiment into the `periodOfOscillation` parameter. If you are following the GUI that I created, be careful that the plot interval is actually not the same as the interval at which the Arduino loops. As such, when you determine the time taken between 2 points, you have to use the plot interval multiplied by the difference between the 2 indices.

Set $K_d = 1.0$ when you first begin and slowly raise the value of K_p till oscillations start to occur when you give a different step size. I used -20° to 20° step for this but you can actually use any reasonable values. You will see that the jump upwards shows a different response than the jump downwards. This is because our propeller can only spin in a single direction, therefore, the control system does not function correctly in the reverse direction. Below is a video of what you should be seeing when you do your experiment.

I have taken a screenshot of the oscillatory response below as well to determine the period of oscillation.



Plot Interval = 50ms
[Desired] = 20.0
[Actual] = 20.809234619140625
[Error] = -0.809234619140625

Kp 10.0
Kd 1.0
Ki 0.0
Angle 20

OSCILLATORY RESPONSE

For my case, it looked like it took 11 plot intervals for one cycle of oscillation. That is $0.05s * 11 = 0.55s$. Putting this value as the `periodOfOscillation` in the python script given above (together with the Kp and Kd values), you will get a γ value of 19.61. Now, we have all our unknowns. You can determine the value of β but there is no specific need to do this since β does not appear by itself in the equation.

That's all for the calibration. Let's head to the next section where we implement the different types of controllers.

Proportional (P) Control

A proportional controller is one that takes the error, multiply it by a constant, then add that result to our control input. A proportional controller will have an equation that looks like this:

$$I_{pwm}[n-2] = \frac{\beta}{\gamma} cos(\phi_{desired}) + K_p(err[n-2])$$

where

$err[n-2] = \phi_{desired} - \phi[n-2]$

The first term is needed to get the system to the desired position, and the second term is our proportional controller. If we substitute this equation into our equation of motion (equation 4 in red) and simplify, we will get the following equation:

$$\phi[n] - 2\phi[n-1] + (1 + t^2\gamma K_p)\phi[n-2] = (t^2\gamma K_p)\phi_{desired}$$

Considering only the homogeneous equation,

$$\phi[n] - 2\phi[n-1] + (1 + t^2\gamma K_p)\phi[n-2] = 0$$

In order to solve for the natural frequencies of the system, we can convert the above to a polynomial equation and solve for the roots of the equation. The polynomial is given by:

$$\lambda^2 - 2\lambda + (1 + t^2\gamma K_p) = 0$$

Solving for the roots of the second order polynomial using the magic equation,

$$\lambda = \frac{2 \pm \sqrt{4 - 4(1 + t^2\gamma K_P)}}{2(1)}$$

Since $t > 0$, $\gamma > 0$, and $K_p > 0$,

$$4 - 4(1 + t^2\gamma K_P) < 1$$

Therefore,

$$\lambda = 1 \pm \sqrt{1 - 1 - t^2\gamma K_p}$$

$$\lambda = 1 \pm t^2\gamma K_p i$$

Here, since $|\lambda| > 1$, the system is unstable for any values of K_p. Recall from the previous chapter that if any of the roots have a magnitude of more than 1, the system response will grow without bound thus it is unstable. In the case of the copter arm control, the copter will start swinging in a larger and larger arc as time passes. This shows that a proportional only controller is insufficient to control a second order system (when you have a second order derivative present in the equation) such as the Copter Angle Control. On the other hand, for a simple first order system such as the motor speed controller in the previous chapter, a proportional controller is sufficient to control the system.

Proportional and Derivative (PD) Control

Since the proportional only controller does not work, we have to think of something else. You will see that once we add the derivative term into the control, magic happens and we can stabilize the system.

Let our PD controller be as follows:

$$I_{pwm}[n-2] = \frac{\beta}{\gamma}cos(\phi_{desired}) + K_p(err[n-2]) + K_d\left(\frac{err[n-2] - err[n-3]}{t}\right) \text{ ———- (5)}$$

This is actually equation 5 from the calibration section. Substituting this into the equation of motion (equation 4 in red),

$$\phi[n] - 2\phi[n-1] + (1 + t^2\gamma(K_p + \frac{K_d}{t}))\phi[n-2] - t\gamma K_d\phi[n-3] = (t^2\gamma K_p)\phi_{desired} \text{ ———- } (6)$$

Considering only the homogeneous equation,

$$\phi[n] - 2\phi[n-1] + (1 + t^2\gamma(K_p + \frac{K_d}{t}))\phi[n-2] - t\gamma K_d\phi[n-3] = 0$$

We can then solve for the natural frequencies of the system by converting the above to a polynomial and solving for the roots of the equation. The polynomial is given by:

$$\lambda^3 - 2\lambda^2 + (1 + t^2\gamma(K_p + \frac{K_d}{t}))\lambda - t\gamma K_d = 0$$

This is a 3rd order polynomial in the form of:

$$a\lambda^3 + b\lambda^2 + c\lambda + d = 0$$

where

$$a = 1$$
$$b = -2$$
$$c = 1 + t^2\gamma(K_p + \frac{K_d}{t})$$
$$d = -t\gamma K_d$$

In order for the system to be stable, none of the 3 roots in the above equation should have an absolute value of more than 1. Since we now have the value of γ through our calibration, we can simply plug in values into the equation and use a polynomial solver such as **this online one** to solve for the roots. For example, in the video above, the γ value is 19.61, $K_p = 10.0$ and $K_d = 1.0$. After plugging in these values and solving the polynomial equation, the 3 roots are:

$$\lambda_1 = 0.02001$$
$$\lambda_2 = 0.99000 + 0.01000i$$
$$\lambda_3 = 0.99000 - 0.01000i$$

If you calculate the magnitude of the roots above, they are all less than 1, albeit being extremely close to 1. This is expected because in the calibration section, I increased the value of K_p to the point of instability. This is also shown in the sustained oscillations in the video (the oscillations take a long time to die down).

So, we now know that a PD controller can stabilize a second order system, but what about the steady-state error? In order to analyze the steady-state error, we will need to do the time infinity analysis. Given that the system is stable, at time infinity,

$$\phi[n-3] = \phi[n-2] = \phi[n-1] = \phi[n] = \phi[\infty]$$

Therefore, equation 6 turns into the following

$$\phi[\infty] - 2\phi[\infty] + (1 + t^2\gamma(K_p + \frac{K_d}{t}))\phi[\infty] - t\gamma K_d\phi[\infty] = (t^2\gamma K_p)\phi_{desired}$$

Simplifying,

$$\phi[\infty] = \phi_{desired}$$

This tells us that the controller will eventually bring the copter arm to out set point. In other words, steady-state error is 0!

Everything looks good now but actually, there is one caveat here. What if there are un-modeled forces that are acting on our copter arm, such as a strong continuous wind? This will add another term on the right hand side of equation 6 so $\phi[\infty]$ will not just be equals to $\phi_{desired}$. This means that there will be a steady-state error in the event there is some external un-modeled forces which there will be, because we cannot account for everything factor in the universe. In fact, the values of γ and β were simply estimates so they are not exact. This in turn means that the $\frac{\beta}{\gamma}cos(\phi_{desired})$ term in our controller will not exactly cancel out with the cosine term in the equation of motion (equation 4 in red).

Is it then possible to create a control system that is versatile enough to adapt to un-modeled forces? The answer is yes! We can achieve it by adding an integral term to our current controller. Let us move on to the next section for the PID controller.

Proportional, Integral and Derivative (PID) Control

As written in the previous section, the integral term helps us counter un-modeled forces and inaccuracies in our model. Let our PID controller be as follows:

$$I_{pwm}[n-2] = \frac{\beta}{\gamma}cos(\phi_{desired}) + K_p(err[n-2]) + K_d\left(\frac{err[n-2] - err[n-3]}{t}\right) + K_iS[n-2] \text{ ———- (7)}$$

where
$S[n] = S[n-1] + t(err[n])$

To put it simply, $S[n]$ is the area under the error time graph. Our controller input has turned out pretty long but the terms that make up it are not difficult to understand at all. The first term is actually the feed-forward function (new term here but it simply means that the function is used to approximate the input for the desired output). The 2nd, 3rd, and 4th terms are the Proportional, Derivative and Integral Controllers respectively.

Due to the summation term $(S[n-2])$ in our controller, we have to do some manipulation first before substituting it into the equation of motion (equation 4 in red). This will be a mess but lets push on!

$$\phi[n] - 2\phi[n-1] + \phi[n-2] = t^2\gamma I_{pwm}[n-2] - t^2\beta cos(\phi[n-2]) \text{ ———- (4)}$$

In equation 4, replace n with $n-1$ to get the following equation:

$$\phi[n-1] - 2\phi[n-2] + \phi[n-3] = t^2\gamma I_{pwm}[n-3] - t^2\beta cos(\phi[n-3]) \text{ ———- (8)}$$

With equation (4) minus equation (8),

$$\phi[n] - 3\phi[n-1] + 3\phi[n-2] - \phi[n-3] = t^2\gamma(I_{pwm}[n-2] - I_{pwm}[n-3]) - t^2\beta(cos(\phi[n-2]) - cos(\phi[n-3])) \text{ ———- (9)}$$

Now, evaluating just the 2 controller (I_{pwm}) terms using equation 7,

$$I_{pwm}[n-2] - I_{pwm}[n-3] = a_1 + b_1 + c_1 + d_1 \text{ ———- (10)}$$

where
$a_1 = K_p(err[n-2] - err[n-3])$
$b_1 = \frac{K_d}{Nt}(err[n-2] - err[n-2-N] - err[n-3] + err[n-3-N])$
$c_1 = K_i(S[n-2] - S[n-3]) = t(err[n-2])$ since $S[n] - S[n-1] = t(err[n])$
$d_1 = \frac{\beta}{\gamma}(cos(\phi_{desired}) - cos(\phi_{desired})) = 0$

and N is the number of time intervals used for calculating the derivative term.

Substituting equation 10 into equation 9,

$$\phi[n] - 3\phi[n-1] + 3\phi[n-2] - \phi[n-3] = a_2 + b_2 + c_2 + d_2 \text{ ———- (11)}$$

where
$a_2 = t^2\gamma K_p(err[n-2] - err[n-3])$
$b_2 = \frac{t\gamma K_d}{Nt}(err[n-2] - err[n-2-N] - err[n-3] + err[n-3-N])$
$c_2 = t^3\gamma K_i(err[n-2])$
$d_2 = -t^2\beta(cos(\phi[n-2]) - cos(\phi[n-3]))$

Here, $err[n] = \phi_{desired} - \phi[n]$

Alright, we are finally done! Now, let us do a time infinity analysis once again. We will start off with the err term first

$$err[n] - err[n-1] = \phi_{desired} - \phi[n] - \phi_{desired} + \phi[n-1]$$

$$err[n] - err[n-1] = -\phi[n] + \phi[n-1]$$

Assuming that the system is stable, at time infinity,

$$\phi[n-3] = \phi[n-2] = \phi[n-1] = \phi[n] = \phi[\infty]$$

Therefore,

$$err[n] - err[n-1] = -\phi[\infty] + \phi[\infty]$$

Thus we can further conclude that

$$err[n] - err[n-1] = 0$$

With this, take a look at equation 11 again and you will realize that $a_2 = 0$ and $b_2 = 0$.

$$d_2 = -t^2\beta(cos(\phi[\infty]) - cos(\phi[\infty])) = 0$$

Looking at d_2, you should notice that the cosine terms cancel away. These are the terms that our controller have to cancel away to reach a certain angle but they cancelled themselves out now! The point here is that now, even if the values of our β and γ are inaccurate, it doesn't really matter because the cosine term in the equation of motion are cancelled out by themselves! Isn't it just amazing? O.O Look at the amount of information available from all these equations and you should feel amazed too!

Anyway, moving on, after doing all the suitable substitutions, at time infinity, the equation of motion reduces to:

$$0 = t^3\gamma K_i(err[\infty])$$

Since$err[\infty] = \phi_{desired} - \phi[\infty]$, the above equation further reduces to:

$$\phi[\infty] = \phi_{desired}$$

This tells us that eventually, the angle of the copter will converge to the angle that we desire so that the steady-state error is 0.

We are now ready to play with the PID controller! There are many ways to tune the values for PID and there is no right or wrong here. **Here's** a good way which is similar to how I determined my PID values. Below is a video of the final outcome! Look at just how versatile it is in rejecting disturbances!

After watching the video, you should understand why this chapter is called relative control. That is because the copter arm stays at the specified angle relative to the box regardless of how the box turns. This is shown in the video at time 1:11. In the next chapter, we will be looking at absolute control, meaning that the copter arm will stay at the specified angle relative to the floor (world) regardless of how the box turns.

On an ending note, if you took a look at the source code for the Arduino, you should notice that there is a `maxSigmaErr` term. This term sets the limit for the integral error. This is needed or else the integral error will get extremely huge when an external disturbance is acting on the system for a sustained period of time. In such an event, if the external disturbance suddenly disappears, the system will need to counter a large error (this will take some time) before it can respond accurately again.

For the copter, if you set the desired angle to 0° and hold the arm up to 30° for an extended period of time, the integral error will slowly get more and more negative. If you do not cap this at some value, when you let go of the copter arm, the propeller will not spin at all at first because of the huge integral error that the PD error has to correct.

Alright! That’s all for this chapter folks! I hope this post has helped you in one way or another! 🙂 If you find any mistakes or have any questions, please feel free to write in the comments section below!

Go to Next Section

PREVIOUS POST
Motor Speed Control

NEXT POST
Copter Angle Control (Absolute)

POSTED IN FEEDBACK CONTROL SYSTEM

8 comments on "*Copter Angle Control (Relative)*"

Matan
December 12, 2018 at 11:05 pm

Hello,
Very extensive and interesting!
I really appreciate the work you put into this project, especially the documenting :).
I've been using Simulink via Matlab support package for Arduino to construct controllers for my Arduino projects.
Recently I decided I'd rather not rely on a paid license, so I wanted to look for a new, free method to design controllers with my Arduino but didn't want to use an Arduino specific software, so I was thinking of Python.

You used Python for the GUI, and Arduino software for the PID controller.
Why didn't you use Python to code the PID controller?
Is Python less ideal for designing controllers for Arduino? and wouldn't it be more convenient to use just one software and not mix the syntaxes?

Thanks

rikisenia.L
December 13, 2018 at 12:45 am

Hi Matan,

Thanks for reading through my post.

The reason for the implementation in this post is actually more personal than practical. I am actually more interested in embedded systems so I want my Arduino to be capable of controlling the system without a computer.
The Python plotting interface is actually for debugging and studying purposes, and the GUI is simply a substitute for a physical controller. For example, for my implementation, I could possibly use a physical controller (joystick etc) to send the desired angle for the arm to the Arduino.

However, do take note that there is a maximum (readily usable) baudrate for Arduino's serial transmission. This means that if you need to send a lot of data to your controller in Python, the rate of data transfer might not be fast enough and some data points might be discarded by the data transmission protocol. That being said, when the calculations gets more complicated, it would be better to move the calculations to the computer, or the other choice is to get a "faster" embedded system.

Matan

December 13, 2018 at 4:46 pm

Got it, that cleared things up for me.

Thanks!

By the way, I also did a pendulum angle control project using a rotor:

https://youtu.be/5A7lLxkIl9A

I looked for similar projects for reference while I was making it but didn't find anything nearly as well documented as your project.

I'd be happy to hear any notes you might have to give me about where I might have done better.
I also tried to make it as low cost as possible.

Thanks

rikisenia.L

December 15, 2018 at 2:36 pm

Hi Matan,

I just saw your project on youtube.
It reminded me of my school days where I used to refer to the same web page for references on control systems too.
Great job on the project!

Aeq
December 3, 2019 at 11:21 am

Hi, thanks for your blog, so much! I think I noticed a typo:

From equation 1 above, let $\dot{\phi}=\theta$ so $\ddot{\phi}=\dot{\theta}$. Then,

$Jr\dot{\phi}=F-mg\cos(\phi)$

shouldn't it be $Jr\dot{\theta}=F-mg\cos(\phi)$?
The $r\dot{\phi}$ is there up to the difference form of the equation.

Thanks

Aeq
December 3, 2019 at 11:44 am

And another typo:

Therefore equation 4 turns into:

$\phi[\infty]-2\phi[\infty]-\phi[\infty]=t2yIpwm[\infty]-t2\beta\cos(\phi[\infty])$

but it seems that it should be:

$\phi[\infty]-2\phi[\infty]+\phi[\infty]=t2yIpwm[\infty]-t2\beta\cos(\phi[\infty])$

Insanoff
February 4, 2020 at 11:27 pm

Phenomenal work! So educational. Thank you sharing your knowledge with us. I feel obligated to try it myself.

Celeste
October 30, 2020 at 3:17 pm

How would it be if I want to graph the angle of inclination of the arm, the duty cycle of the motor pwm signal and the position to be controlled?

Comments are closed.

Categories

Select Category

Archives

September 2019 (3)

February 2019 (1)

January 2019 (1)

September 2018 (1)

July 2018 (4)

March 2018 (3)

February 2018 (5)

November 2017 (6)

July 2017 (2)

April 2017 (6)

March 2017 (4)

February 2017 (2)

January 2017 (9)

December 2016 (24)

November 2016 (1)

About | Life | Projects | Woodcarving | Contact